UNIVERSITY OF MARYLAND

COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

1 LEVEL II

9 Technical rept., 12 29

14 CSC -TR-732 / 11 Feb[redacted]79

DAAG-53-76C-0138,

15 DARPA Order -3206

6 REGION REPRESENTATION: BOUNDARY CODES FROM QUADTREES.

10 Charles R./Dyer,
Azriel/Rosenfeld
Hanan/Samet

Computer Science Center and
Computer Science Department
University of Maryland
College Park, MD 20742

ABSTRACT

There has been recent interest in the use of quadtrees to represent regions in an image. It thus becomes desirable to develop efficient methods of conversion between quadtrees and other types of region representations. This paper presents an algorithm for converting from quadtrees to a simple class of boundary codes.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA order 3206) is gratefully acknowledged, as is the help of Ms. Kathryn Riley in preparing this paper.

403018

[illegible]

3.1 Circular Model

We will first compute f_0 and f_1 for circles.

$\underline{f_0}$

Considering a chord passing through a point at a distance r from the center of a circle of radius R, $r \leq R$ (Fig. 5), we have

$$\Pr(Z \leq z) = \frac{1}{\pi R^2} \int_{\sqrt{R^2 - \frac{z^2}{4}}}^{R} 2\pi r \, dr \left\{ \frac{2}{\pi} \int_{\sin^{-1}\left(\frac{1}{r}\sqrt{R^2 - \frac{z^2}{4}}\right)}^{\pi/2} d\theta \right\}$$

$$= \frac{4}{\pi R^2} \int_{\sqrt{R^2 - \frac{z^2}{4}}}^{R} r \left\{ \frac{\pi}{2} - \sin^{-1}\left(\frac{1}{r}\sqrt{R^2 - \frac{z^2}{4}}\right) \right\} dr$$

$$= \frac{4}{\pi R^2} \left[\frac{\pi z^2}{16} - \int_{\sqrt{R^2 - \frac{z^2}{4}}}^{R} r \sin^{-1}\left(\frac{1}{r}\sqrt{R^2 - \frac{z^2}{4}}\right) dr \right]$$

$$= \frac{z^2}{4R^2} - \frac{4}{\pi R^2} \int_{\sqrt{R^2 - \frac{z^2}{4}}}^{R} r \sin^{-1}\left(\frac{1}{r}\sqrt{R^2 - \frac{z^2}{4}}\right) dr$$

$$= \frac{z^2}{4R^2} - \frac{(4R^2 - z^2)}{\pi R} \int_{1}^{\frac{R}{\sqrt{R^2 - \frac{z^2}{4}}}} u \sin^{-1}\left(\frac{1}{u}\right) du$$

Let $I = \int u \sin^{-1}\left(\frac{1}{u}\right) du$

Substituting $v = \frac{1}{u}$, we have

This type of boundary representation is called a <u>chain code</u>. Generalized chain codes, involving more than four directions, can also be used. Chain codes provide a very compact region representation, and make it easy to detect features of the region boundary, such as sharp turns ("corners") or concavities. On the other hand, it is harder to determine properties such as elongatedness from a chain code, and it is also difficult to perform operations such as union and intersection on regions represented by chain codes. A general introduction to chain codes and their uses can be found in [1].

Another class of region representations involves various types of maximal "blocks" that are contained in a given region. For example, we can represent a region R as a linked list of the runs (of pixels) in which R meets the successive rows of the array [2]. Here each "block" is a 1-by-m rectangle, where m is the run length; the runs are the largest such blocks that R contains, and R is determined by specifying the initial points (or centers) and lengths of the runs. Alternatively, we can represent R by the set of maximal square blocks (or blocks of any other desired shape) that it contains; here R is determined by specifying the centers and radii of these blocks. This representation is called the <u>medial axis transformation</u>, or MAT [3]. It is somewhat less compact than chain code [4], but it has advantages with respect to performing union and intersection operations or detecting properties such as elongatedness (in terms of the smallness of the radii relative to the number of centers).

There has been recent interest in an approach to region representation based on successive subdivision of the array into quadrants. If the region does not cover the entire array, we subdivide the array, and repeat this process for each quadrant, each subquadrant,... as long as necessary, until we obtain blocks (possibly single pixels) that are entirely contained in the region or entirely disjoint from it. The resulting blocks for the region of Figure 1a are shown in Figure 1b. This process can be represented by a tree of degree 4 (for brevity: a quadtree) in which the entire array is the root node, the four sons of a node are its quadrants, and the leaf nodes correspond to those blocks for which no further subdivision is necessary.* The quadtree representation for Figure 1b is shown in Figure 1c. Note that here again we are representing the region as a union of maximal blocks, but this time the blocks must have standard sizes and positions (powers of 2). Since the array was assumed to be 2^n-by-2^n, the tree height is at most n. This method of region representation was proposed by Klinger [6-7]; it has also been used for image representation (e.g., [8-11]). It is relatively compact, and is also well suited to operations such as union and intersection, and to detecting various region properties. A

* The quadtree region representation described here should not be confused with the quadtree representation of two-dimensional point data introduced by Finkel and Bentley [5].

recent Ph.D. thesis by Hunter [12] in the domain of computer graphics develops a variety of algorithms for the manipulation of quadtree region representations. Those algorithms, however, allow a node to store the list of coordinate points that describe the polygon from which the quadtree was constructed.

Since the quadtree and border representations both have computational advantages, it is of interest to develop methods of converting from one representation to the other. We shall now present an algorithm for deriving a clockwise boundary code from the quadtree representation of a given region.

2. Definitions and notation

Let each node in a quadtree be stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SW, and SE. These items will be referenced as FATHER(P), NWSON(P), NESON(P), SWSON(P), and SESON(P), respectively, for a given node P. The sixth field, named NODETYPE, describes the contents of the block of the array which the node represents, i.e., WHITE if the block contains no pixels in the region, BLACK if the block contains only pixels in the region, and GRAY if it contains pixels of both types.

Let the four sides of a node's block be called its N, S, E, and W sides. Two nodes are said to be adjacent along the N side of the first, for example, if the pair of blocks represented by these nodes touch along that side (not just at a corner). Given this notation for a node's sides and quadrants, as illustrated in Figure 2, we now define several functions which conveniently describe the geometry of these labelings. Throughout this paper, we use T and U as side variables, K and L as quadrant variables, and P, Q, X, Y, and Z as node variables.

Define CSIDE(T) to be the side which is adjacent to side T in the clockwise direction, e.g., CSIDE('N') = 'E'. Similarly, define CCSIDE(T) and OPSIDE(T) to be the sides which are adjacent in the counterclockwise direction and on the opposite side from T, respectively. For example, CCSIDE('N') = 'W' and OPSIDE('N') = 'S'. The value of the boolean function ADJ(T,K) is true if and only if quadrant K is adjacent to side T of the node's block, e.g., ADJ('N','NW') = true.

Define REFLECT(T,K) to be the quadrant in which quadrant K lies after being reflected about the side T axis. For example, REFLECT('W','NE') = 'NW' and REFLECT('N','NW') = 'SW'. QUAD(T,U) is defined to be the quadrant which touches the corner formed by sides T and U (if T and U are opposite sides, the function is undefined), e.g., QUAD('E','N') = 'NE'. LINK(T) returns the chain code direction associated with side T. If we are using the coding described in Section 1, this implies LINK('N') = 0, LINK('W') = 1, LINK('S') = 2, and LINK('E') = 3.

It is also convenient to be able to access certain properties of a node concisely. Therefore, define SONTYPE(P) to be the quadrant type of P relative to P's father, e.g., SONTYPE(P) = 'NW' iff NWSON(FATHER(P)) = P. Let SON(P,K) be the Kth son of node P, where K is a quadrant. For example, SON(P,'NW') = NWSON(P). Finally, define DEPTH(P) to be the length of the path from node P to the root of the quadtree, TREEDEPTH to be the maximum depth of any node in the quadtree, and LEVEL(P) to be TREEDEPTH-DEPTH(P).

2. Definitions and notation

Let each node in a quadtree be stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SW, and SE. These items will be referenced as FATHER(P), NWSON(P), NESON(P), SWSON(P), and SESON(P), respectively, for a given node P. The sixth field, named NODETYPE, describes the contents of the block of the array which the node represents, i.e., WHITE if the block contains no pixels in the region, BLACK if the block contains only pixels in the region, and GRAY if it contains pixels of both types.

Let the four sides of a node's block be called its N, S, E, and W sides. Two nodes are said to be adjacent along the N side of the first, for example, if the pair of blocks represented by these nodes touch along that side (not just at a corner). Given this notation for a node's sides and quadrants, as illustrated in Figure 2, we now define several functions which conveniently describe the geometry of these labelings. Throughout this paper, we use T and U as side variables, K and L as quadrant variables, and P, Q, X, Y, and Z as node variables.

Define CSIDE(T) to be the side which is adjacent to side T in the clockwise direction, e.g., CSIDE('N') = 'E'. Similarly, define CCSIDE(T) and OPSIDE(T) to be the sides which are adjacent in the counterclockwise direction and on the opposite side from T, respectively. For example, CCSIDE('N') = 'W' and OPSIDE('N') = 'S'. The value of the boolean function ADJ(T,K) is true if

For example, the three cases shown in Figure 3 are treated as follows:

a) In this case X cannot extend beyond P, though it may be bigger than Q. If X is black, the new pair is (X,Q), the boundary turns left, and the new link is the west side of Q (if X is larger than Q) or the east side of X (otherwise). If X is white, the new pair is (P,X), the boundary does not turn, and the new link is the north side of X. See Figure 4a-b.

b) In this case X cannot extend beyond Q, though it may be bigger than P. If X is black, the new pair is (X,Q), the boundary does not turn, and the new link is the south side of X. If X is white, the new pair is (P,X), the boundary turns right, and the new link is the west side of P (if P is no larger than X) or the east side of X (otherwise). See Figure 4c-d.

c) Assume that the region is 4-connected, so that blocks touching only at a corner are not regarded as adjacent. Then if both X and Y are black (they need not be distinct nodes), the new pair is (Y,Q), the boundary turns left, and the new link is the east side of Y(if X and Y are distinct and Y is no bigger than Q) or the west side of Q (otherwise; note that if X and Y are the same node, Y must extend at least to the end of Q because neighboring nodes can't properly overlap.) If X is black and Y is white, the new pair is (X,Y), the boundary does not turn, and the link is the south side of X (if X is no larger than Y) or the north side of Y (otherwise). If X is white, the new pair is (P,X) regardless of the node type of Y by virtue of the 4-connectedness property. The new link is the west side of P (if P is no bigger than X) or the east side of X (otherwise); the boundary turns right. See Figure 4e-i. Alternatively, we could specify a similar set of rules if we considered the region to be 8-connected.

4. Formal statement of the algorithm

The following ALGOL-like procedures specify the complete boundary following algorithm. The main program finds and marks the initial (black, white) pair of nodes. NEXT_LINK is the recursive algorithm which outputs the link associated with the current pair, and then finds the next pair. Algorithm FIND_ NEIGHBOR is used to find the adjacent nodes X and Y. OVERLAP determines which of the three cases shown in Figure 3 applies to the current pair.

```
begin
comment  given quadtree QUADTREE, find a black node P which
         is on the region boundary and has no black nodes adjacent
         to its north side.  Q, a white node, is the western-most
         of P's northern neighbors.  The pair (P,Q) defines the
         initial chain segment;
node  P,Q;
P ← ROOT (QUADTREE);
while  NODETYPE(P) = 'GRAY' do
     if  NODETYPE (NWSON(P)) ≠ 'WHITE'
          then  P←NWSON(P)
          else if NODETYPE (NESON(P)) ≠ 'WHITE'
                 then P←NESON (P)
                 else if NODETYPE (SWSON(P)) ≠ 'WHITE'
                        then P←SWSON(P)
                        else P←SESON(P);
Q ← FIND_NEIGHBOR (P,'N','W');
Mark (P,Q) as starting pair;
NEXT_LINK (P,Q,'N')
end
```

```
procedure  NEXT_LINK (P,Q,T):
begin
comment  given adjacent leaf nodes P, Q, where P is black, Q is
         white, and side T of P touches Q, output the chain
         description for this section of the boundary;
node P,Q,X,Y,Z; side T; integer i;
if (P,Q) is the marked starting pair and we've seen it
   before then halt;

for i←1 step 1 until 2^MIN(LEVEL(P),LEVEL(Q))
    do print (LINK(T));

comment  determine next pair of nodes and their common side
         and recursively call NEXT_LINK;
Z←OVERLAP (P,Q, CSIDE(T));
if Z=P
    then begin
        comment Black overlaps white;
        X←FIND_NEIGHBOR (Q, CSIDE(T), OPSIDE(T));
        if NODETYPE(X) = 'WHITE'
            then NEXT_LINK (P,X,T)              /*Figure 4a*/
            else NEXT_LINK (X,Q, CCSIDE(T))     /*Figure 4b*/
        end
        else if Z=Q
            then begin
                comment  white overlaps black;
                X←FIND_NEIGHBOR (P, CSIDE(T),T);
                if NODETYPE(X) = 'WHITE'
                    then NEXT_LINK (P,X, CSIDE(T))    /*Figure 4c*/
                    else NEXT_LINK (X,Q,T)            /*Figure 4d*/
                end
            else begin
                comment  Black and white aligned;
                X←FIND_NEIGHBOR (P, CSIDE(T),T);
                if NODETYPE(X) = 'WHITE'
                    then NEXT_LINK (P,X, CSIDE(T))    /*Figs. 4e and 4g*/
                    else begin
                        Y←FIND_NEIGHBOR (Q, CCSIDE(T),T);
                        if NODETYPE(Y) = 'BLACK'
                            then NEXT_LINK (Y,Q, CCSIDE(T)) /*Figs. 4f,4i*/
                            else NEXT_LINK (X,Y,T)   /*Figure 4h*/
                        end
                end
    end
```

```
procedure FIND_NEIGHBOR (P,T,U):
begin
comment Given node P, return node Q which is adjacent to side
        T of P and touches P's TU corner;
node  P,Q; side T,U;
Q←P;
STACK←empty
while  ADJ(T,SONTYPE(Q)) do
     begin
     comment find the nearest common ancestor of P and Q;
     push SONTYPE(Q) onto STACK;
     Q←FATHER(Q);
     end;
Q←SON (FATHER(Q), REFLECT (T, SONTYPE(Q)));
while  NODETYPE(Q) = 'GRAY' do
     begin
     comment  follow reflected path back down tree to locate
              neighbor.  If STACK is empty, then Q is smaller than P;
     if  STACK not empty
         then begin  Q←SON (Q, REFLECT(T, top(STACK)));
                     pop STACK
              end
         else  Q←SON (Q, QUAD(OPSIDE(T),U))
     end;
return (Q)
end
```

```
procedure OVERLAP (P,Q,T):
begin
comment given two nodes P and Q which touch along a side U, where
        CSIDE(U)=T, determine whether P extends farther in the
        T direction than Q (return P), Q extends farther than P
        (return Q), or both their Tth sides are aligned
        (return 0);
node P,Q,Lo,Hi; side T; integer Dp, Dq, Diff, i;
Dp = DEPTH(P);
Dq = DEPTH(Q);
if Dp=Dq then return (0);
if MAX(Dp,Dq) = Dq
    then begin Lo←P; Hi←Q end  else begin Lo←Q; Hi←P  end
Diff←|Dp-Dq|
comment The smaller of the two nodes cannot extend farther than
        the other because this would imply that P and Q
        properly overlap, which is impossible.  At best the
        smaller node can be aligned with the other one, and this
        occurs if and only if the smaller node is at the extreme
        T side relative to the nearest common ancestor of P and Q;
for i←1 step 1 until Diff do
    begin
    if ADJ (T,SONTYPE(Lo)) then return (Hi);
    Lo←FATHER (Lo)
    end;
return (0)
end
```

5. An example

Let us consider the quadtree shown in Figure 1. Notice that this quadtree has 57 nodes whereas an array representation would have required a 16 by 16 (=256-cell) logical array. The main program finds the initial pair (19,13) and then calls NEXT_LINK to do the boundary following from this point. Table 1 specifies the arguments of NEXT_LINK at each recursive call along with the link output as a result of this pairing.

Black node P	White node Q	Side T	Link output
19	13	N	0^1
14	13	W	1^1
14	2	N	0^1
14	15	E	3^1
21	15	N	0^1
16	15	W	1^1
16	9	N	0^1
10	9	W	1^1
10	7	N	0^1
10	8	E	3^1
16	8	N	0^2
16	17	E	3^4
29	30	E	3^1
29	34	S	2^1
28	33	S	2^1
27	32	S	2^1
27	26	W	1^1
16	26	S	2^1
25	26	E	3^1
25	31	E	3^1
25	35	E	3^1
38	35	N	0^1
38	39	E	3^1
38	42	S	2^1
25	41	S	2^4
25	24	W	1^4
22	11	W	1^2
22	18	N	0^1
19	18	W	1^1
19	13	N	--

Table 1. Sequence of calls of NEXT_LINK for quadtree in Figure 1. The exponent on the output indicates the number of unit length links associated with the boundary specified by the given (P,Q) pair.

6. Analysis

The speed of this boundary coding algorithm is determined by the procedure NEXT_LINK which is called for each boundary segment associated with a (black, white) adjacent node pair. The time required to output the individual links is proportional to the region's perimeter, where perimeter is defined to be the number of unit-square pixels on the region's boundary (not the number of black nodes which are adjacent to white nodes). The additional cost of the algorithm is determined by the calls to procedures OVERLAP and FIND_NEIGHBOR. Given a pair (P,Q) of adjacent black, white nodes, each procedure is called once in order to find the next pair. The time required is measured by the number of nodes visited and depends on the relative positions of P and Q, and adjacent nodes X and Y in the quadtree.

Procedure OVERLAP takes time proportional to the sum of the depths of nodes P and Q in the quadtree, although this can be reduced to the sum of the path lengths to their nearest common ancestor. Further refinements can be made based on the relationship between P and Q so that OVERLAP doesn't need to be called for each call to NEXT_LINK. For example, having determined that P extends beyond Q and X is smaller than Q (which we can detect while searching for X), then the new pair is related either by case a or b (cf. Figure 4) according to whether X is white or black respectively.

Procedure FIND_NEIGHBOR must be called for each boundary segment defined by a (black,white) node pair. Consider, for example, the call X←FIND_NEIGHBOR(P,T,U). The time required is equal to the sum of the path lengths from nodes P and X to their nearest common ancestor. Thus in the worst case a single call will take height of the quadtree time. We now analyze the average case.

In a complete quadtree of height n there are $2^{2n}-2^n$ leaf nodes which have east neighbors. Of these neighbor pairs 2^n have their nearest common ancestor at level n, $2\cdot 2^n$ at level n-1, $\ldots, 2^{n-i}\cdot 2^n = 2^{2n-i}$ at level i,..., and 2^{2n-1} at level 1. For example, in Figure 5 nodes corresponding to squares 1-8 have eastern neighbors 1'-8' and nearest common ancestors at level 3; nodes for squares 9-24 have eastern neighbors 9'-24' and nearest common ancestor at level 2. (These same frequencies also occur for each of the other sides north, south and west.) Therefore, if we assume that P and X are both in level 0 and P is equally likely to occur at any of the 2^{2n} possible positions, then the average time required by FIND_NEIGHBOR is

$$((2^n\cdot 2n) + (2^{n+1}\cdot 2(n-1)) + \cdots + (2^{2n-i}\cdot 2i) + \cdots + (2^{2n-1}\cdot 2))/(2^{2n}-2^n)$$

$$\approx \sum_{i=1}^{n} 2i/(2^i)$$

$$= (2^{n+1}-(n+2))/2^{n-1}$$

$$< 4$$

That is, P and X's nearest common ancestor is on the average at level 2.

When P and X are both at level k, k>0, a similar argument shows that on the average only a constant amount of time is required for each call to FIND_NEIGHBOR. In general, if P is at level j and X is at level k, then the average time is equal to $|j-k|+4$.

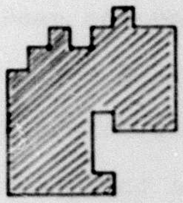
The average total time required by all calls to FIND_NEIGHBOR, and consequently NEXT_LINK, depends on the expected disparity in the sizes at leaf nodes which are adjacent to leaf nodes of the opposite type. If all leaf nodes are at level 0, for example, then the average total time to traverse the entire boundary is O(number of black boundary nodes). Further analysis would require knowledge of expected region shapes and positions and is beyond the scope of this paper.

7. Concluding remarks

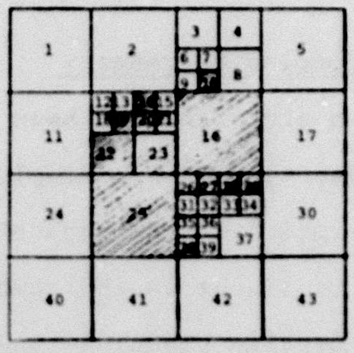
An algorithm has been presented for converting the quadtree representation of a simply-connected region into the chain code description of the region's boundary. The boundary traversal time is linear in the number of boundary nodes in the quadtree when certain plausible assumptions are made about the form of the quadtree.

In the case where a region may have holes, we may extend the algorithm by simply adding a quadtree traversal procedure which systematically visits all black nodes upon completion of the first boundary following sequence. If that scan discovers a black boundary node having a boundary edge which was not marked by the boundary follower, then the scan is temporarily interrupted so that the boundary of which it is a part can be followed.

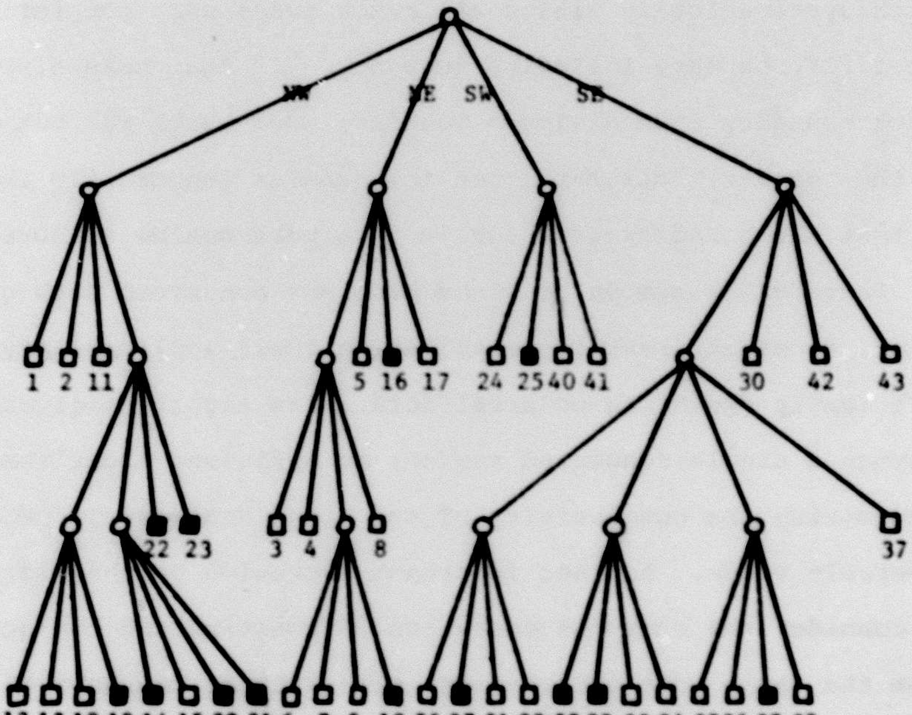
There are a number of other problems concerned with quadtrees which are of interest in establishing their applicability for efficiently operating on areal data. The algorithm given here assumes a single connected region; an efficient algorithm for determining the connectivity of the black nodes would be of considerable value. Another important extension of this work is to consider the converse operation of constructing the quadtree from the chain code of a region; this will be the subject of a subsequent paper.



a. Region.



b. Block decomposition of the region in (a).



c. Quadtree representation of the blocks in (b).

Figure 1. A region, its maximal blocks, and the corresponding quadtree. Blocks in the region are shaded, background blocks are blank.

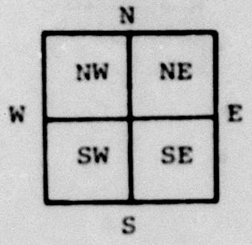


Figure 2. Labeling of a block's four quadrants and four sides.

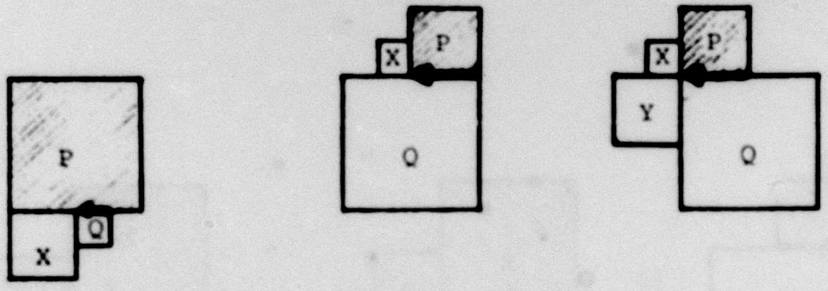


Figure 3. Possible overlap relationships between the (black,white) adjacent node pair (P,Q). The heavy arrow indicates the boundary segment just output.

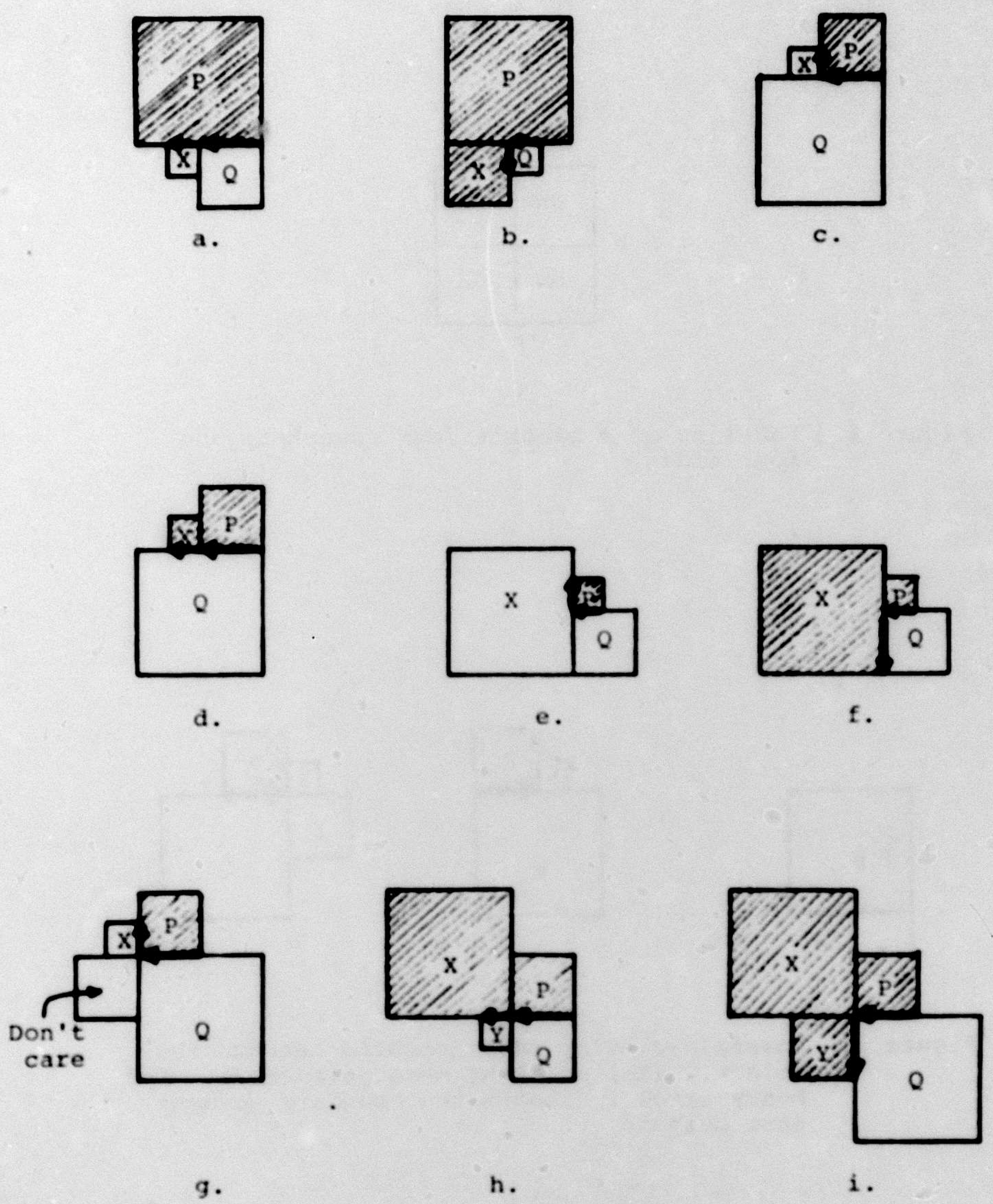


Figure 4. Possible configurations of P, Q and their neighbor blocks in determining the next (black,white) pair. Arrows indicate the boundary segments associated with the old and new pairs.

	9	9'	1	1'	17	17'	
	10	10'	2	2'	18	18'	
	11	11'	3	3'	19	19'	
	12	12'	4	4'	20	20'	
	13	13'	5	5'	21	21'	
	14	14'	6	6'	22	22'	
	15	15'	7	7'	23	23'	
	16	16'	8	8'	24	24'	

Figure 5. Illustration of nearest neighbor computation.

References

1. H. Freeman, Computer processing of line-drawing images, Computing Surveys 6, 1974, 57-97.

2. D. Rutowitz, Data structures for operations on digital images, in G. C. Cheng et al., eds., Pictorial Pattern Recognition, Thompson Book Co., Washington, DC, 1968, 105-133.

3. H. Blum, A transformation for extracting new descriptors of shape, in W. Wathen-Dunn, ed., Models for the Perception of Speech and Visual Form, M.I.T. Press, Cambridge, MA, 1967, 362-380.

4. J. L. Pfaltz and A. Rosenfeld, Computer representation of planar regions by their skeletons, Comm. ACM 10, 1967, 119-122, 125.

5. R. A. Finkel and J. L. Bentley, Quadtrees: a data structure for retrieval on composite keys, Acta Informatica 4, 1974, 1-9.

6. A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics Image Processing 5, 1976, 68-105.

7. N. Alexandridis and A. Klinger, Picture decomposition, tree data-structures, and identifying directional symmetries as node combinations, ibid. 8, 1978, 43-77.

8. S. L. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, ibid. 4, 1975, 104-119.

9. S. L. Horowitz and T. Pavlidis, Picture segmentation by a tree-traversal algorithm, JACM 23, 1976, 368-388.

10. S. L. Tanimoto, Pictorial feature distortion in a pyramid, Computer Graphics Image Processing 5, 1976, 333-352.

11. E. M. Riseman and M. A. Arbib, Computational techniques in the visual segmentation of static scenes, ibid. 6, 1977, 221-276.

12. G. M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REGION REPRESENTATION: BOUNDARY CODES FROM QUADTREES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-732
7. AUTHOR(s) Charles R. Dyer, Azriel Rosenfeld, and Hanan Samet		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Center University of Maryland College Park, MD 20842		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Night Vision Laboratory Ft. Belvoir, VA 22060		12. REPORT DATE February 1979
		13. NUMBER OF PAGES 27
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pattern recognition	Regions
Image processing	Chain coding
Computer graphics	Quadtrees
Cartography	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

There has been recent interest in the use of quadtrees to represent regions in an image. It thus becomes desirable to develop efficient methods of conversion between quadtrees and other types of region representations. This paper presents an algorithm for converting from quadtrees to a simple class of boundary codes.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE